Under the dome: preventing hardware timing information leakage

By **Mathieu Escouteloup** (Inria)
Advisors: **Ronan Lashermes** (Inria)
Christophe Bidan (CentraleSupélec)
Jacques Fournier (CEA-Leti)

November 12, 2021

Table of contents

1. An isolation issue
2. ISA contextualization
3. Shared resources design
4. Timing evaluation
5. Conclusion

Table of contents

1 An isolation issue

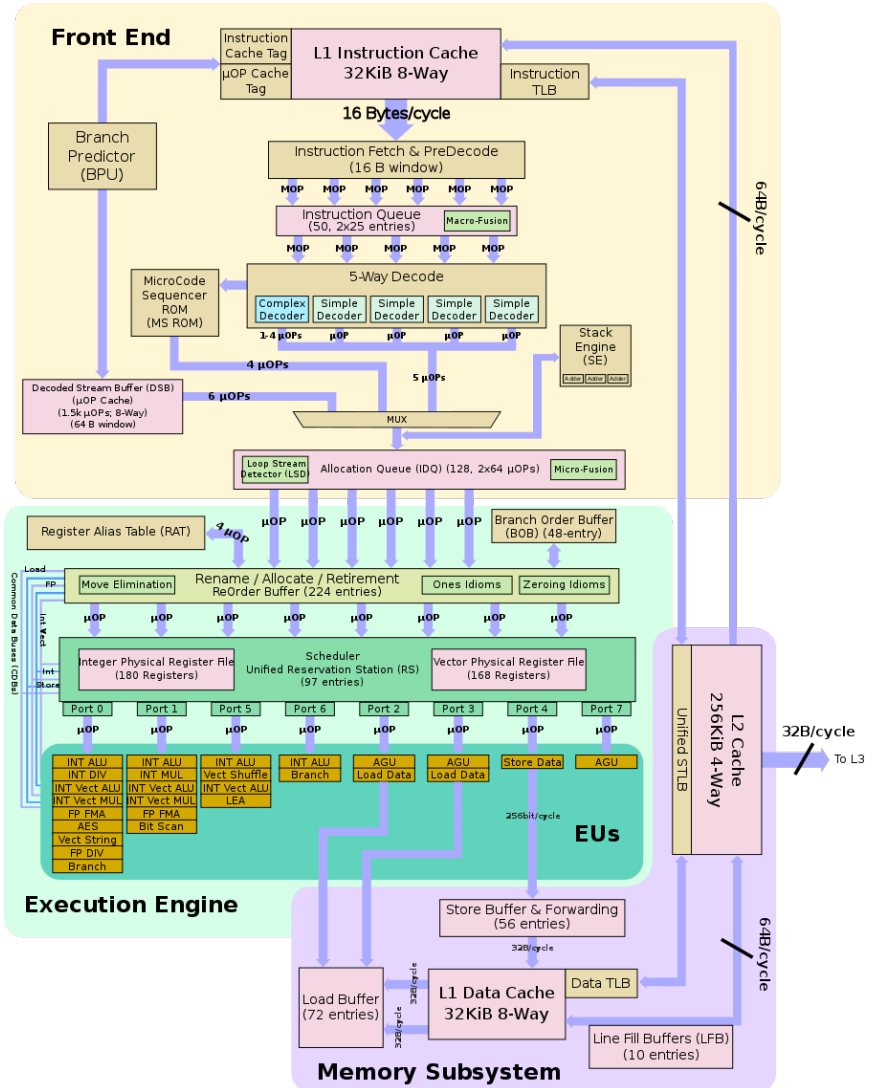
2 ISA contextualization

3 Shared resources design

4 Timing evaluation

5 Conclusion

Microarchitecture: the Intel's Skylake example



Microarchitectural sharing

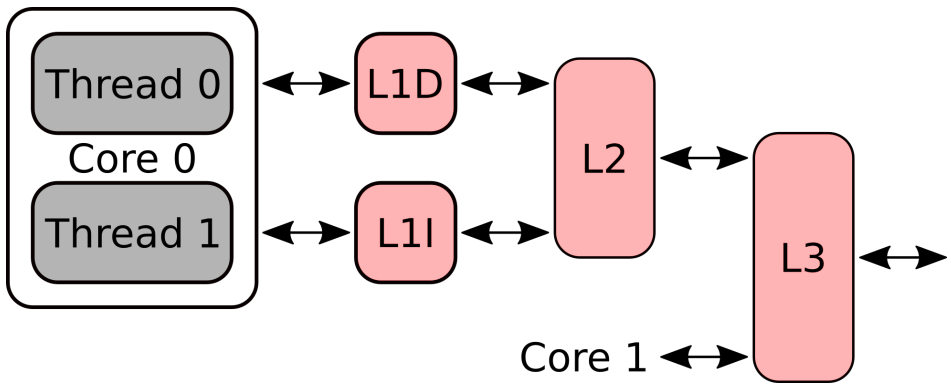
Definition

- Multiple entities can request the same resource.
- Entities: hardware threads, cores, processes ...
- Resources: cache memories, buffers, execution units, buses ...

Two kinds of sharing

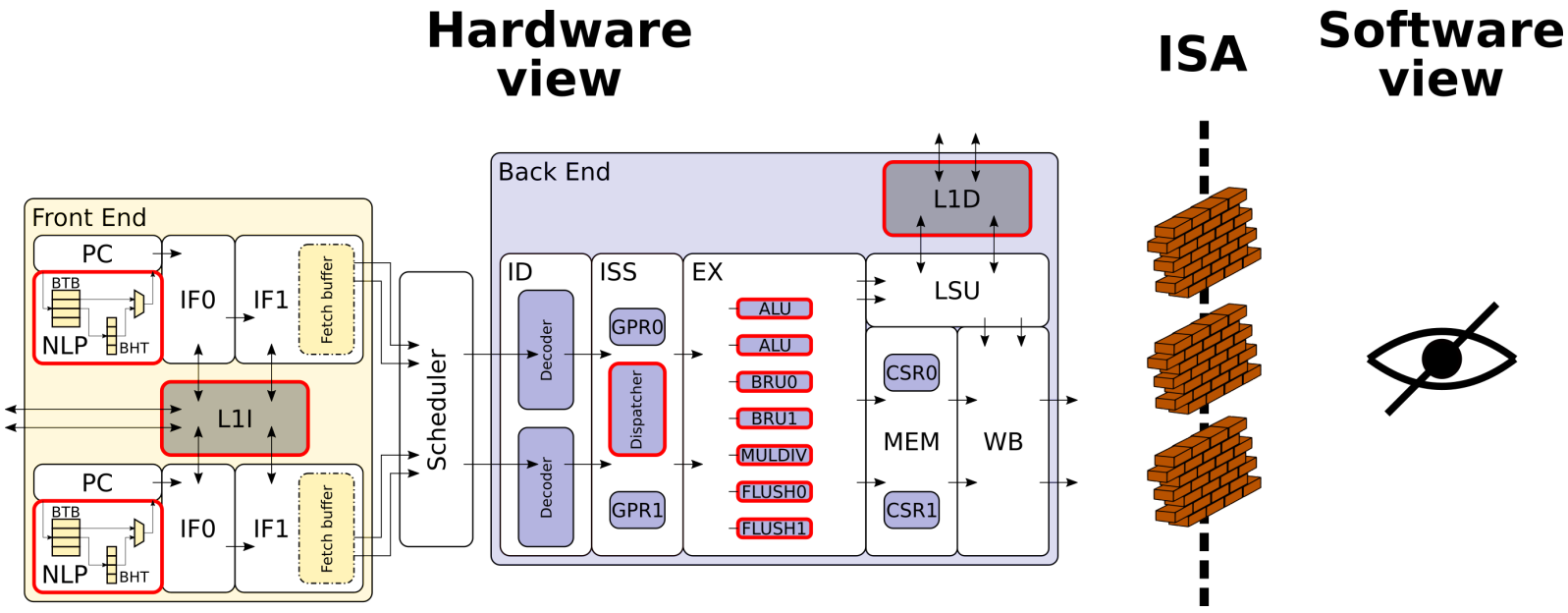
- Temporal: use the same resource but at different points in time.
- Spatial: use the same resource at the same time.
- Both can be combined.

The cache memory example



Data but also timing informations are shared between the entities.

An implementation issue ...



- Targeted shared resources are in the microarchitecture.
- Leakages depend on the implementation.
- Microarchitecture cannot be controlled by the software.

... but not only !

Table of contents

1 An isolation issue

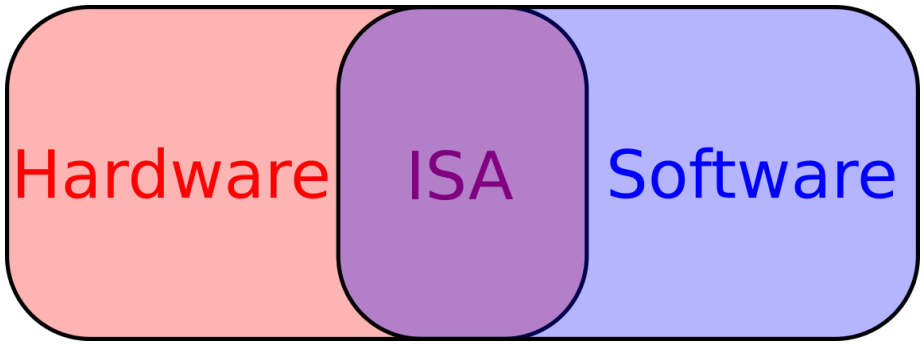
2 ISA contextualization

3 Shared resources design

4 Timing evaluation

5 Conclusion

A global issue



- **Which part knows the application logic ?**
- **Which part can efficiently make the isolation ?**
- **How can they exchange information ?**

The whole system is concerned!

How to modify the ISA ?

Constraints:

1. Consider the whole isolation issue: temporal **and** spatial sharing.
2. Create custom security domains.
3. Usable for simple microcontrollers or complex servers.
4. Preserve the architecture abstraction.

Contextualization

Associate a security domain to each data and resource. Our model: a **dome**.

Our dome model.

New dedicated register:

- identifier: an unique number for each security domain.
- capability: indications on domain's needs.

New instruction: `DOME.SWITCH`.

- Indicates a domain change.
- The hardware manages the shared resources.
- **Successful** → a new domain can be safely executed.

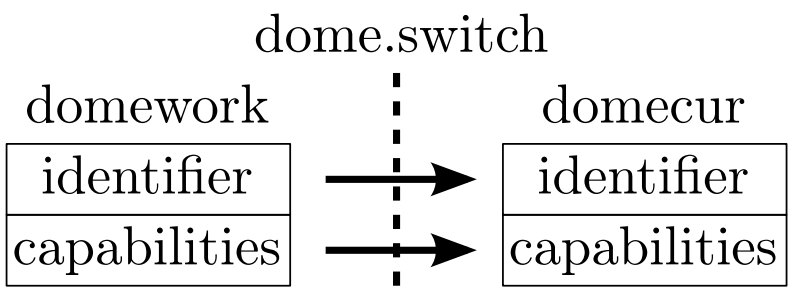


Table of contents

1 An isolation issue

2 ISA contextualization

3 Shared resources design

4 Timing evaluation

5 Conclusion

Our goal

Shared resource security property

The only information that a security domain may extract from a shared resource is the domain's own data or the resource's static availability.

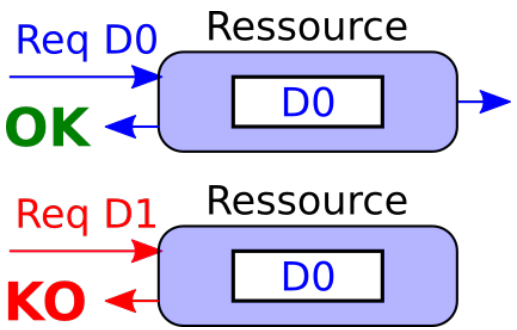
Strategies

- Define generic principles to design secure shared resources.
- 3 complementary strategies: lock, split and flush.

Design strategy: lock

Principle: static allocation.

The different minimal resources needed by a security domain must be allocated during the domain creation and locked until its deletion.

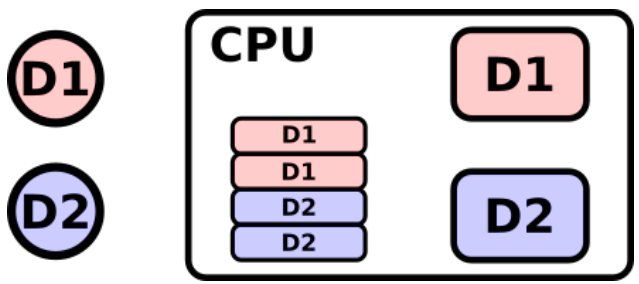


Mechanisms: static allocation and tagged resources.

Design principles: split

Principle: partitioning.

A resource able to handle requests from multiple security domains simultaneously must be able to partition each domain state in its own isolated compartment. States and data cannot be shared.

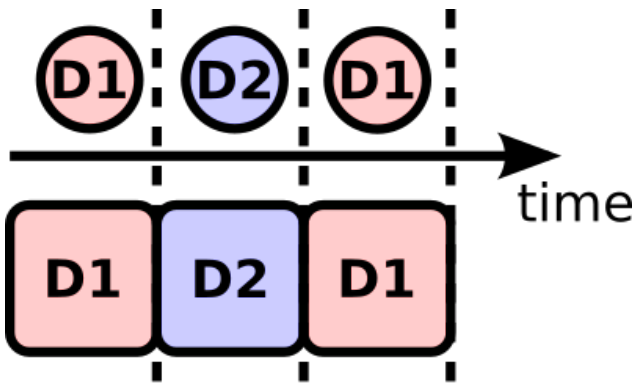


Mechanism: spatial partitioning.

Design principles: split

Principle: availability split.

A spatially shared resource must ensure that, at any given time, its availability for any security domain is independent from the domains being served.

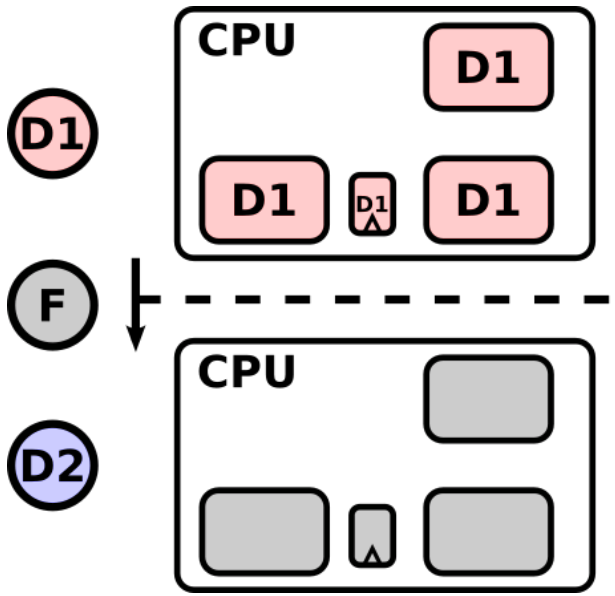


Mechanism: temporal partitioning.

Design principles: flush

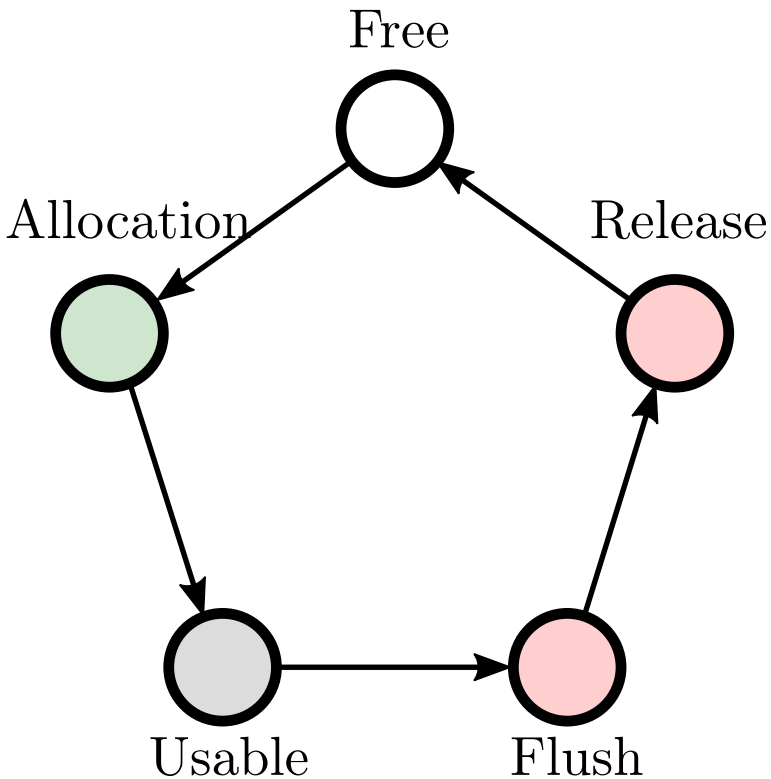
Principle: release.

When a security domain ends, all its associated resources must be released only when all persistent states have also been erased.



Mechanism: flush traces.

Resource lifecycle



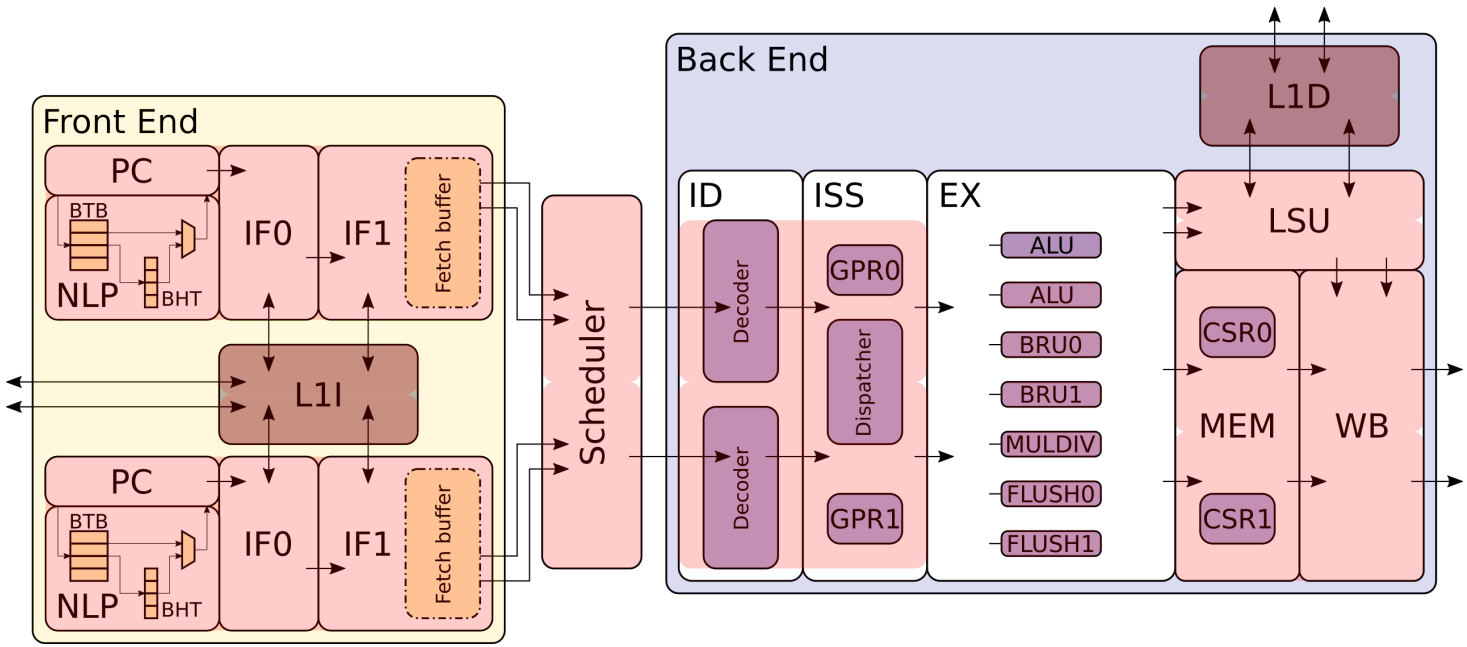
No spatial sharing → **Usable** or **Flush**

Software view.

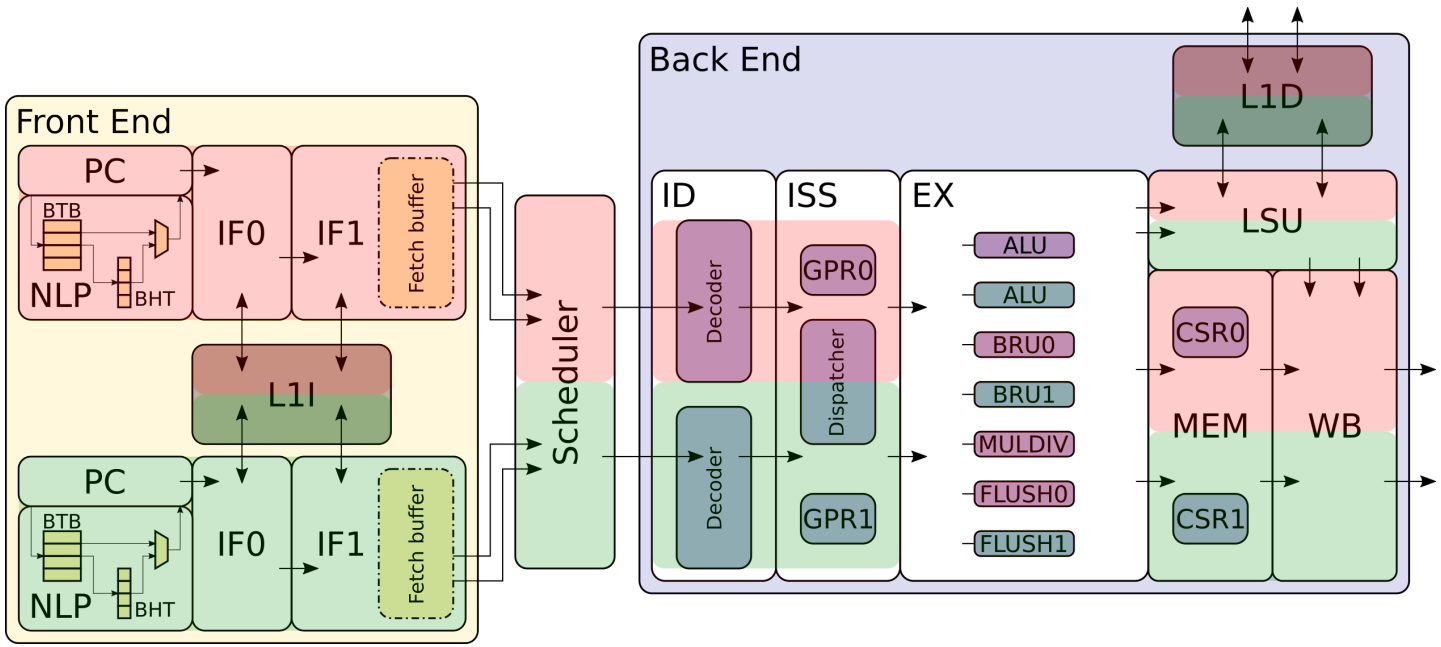
```
1.  # OLD DOME
2.  old-app:
3.    ...
4.    ...
10. switch-code:
11.   csrw nextid,a0    # config
12.   dome.switch a0          # switch


13. # NEW DOME
14. new-app:
15.   ...
16.   ...
```

Hardware view: before switch.



Hardware view: after switch.



Successfully implemented in two cores, one with SMT.

Table of contents

1 An isolation issue

2 ISA contextualization

3 Shared resources design

4 Timing evaluation

5 Conclusion

An implementation agnostic benchmark

Goal:

to quantitatively evaluate information leakages in the microarchitecture.

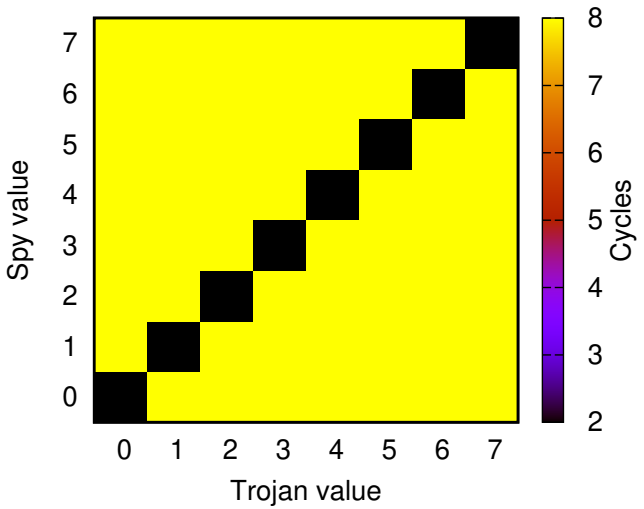
Constraints:

- show the lack of timing information leakages,
- consider common shared resources,
- focus on vulnerability, not exploitability.

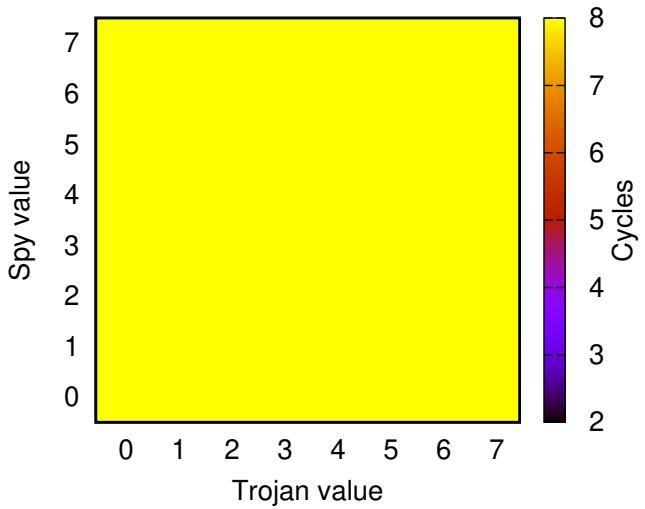
Scenario:

- a trojan encodes a value in a shared resource state,
- a spy tries to recover the value.

The cache example: temporal sharing

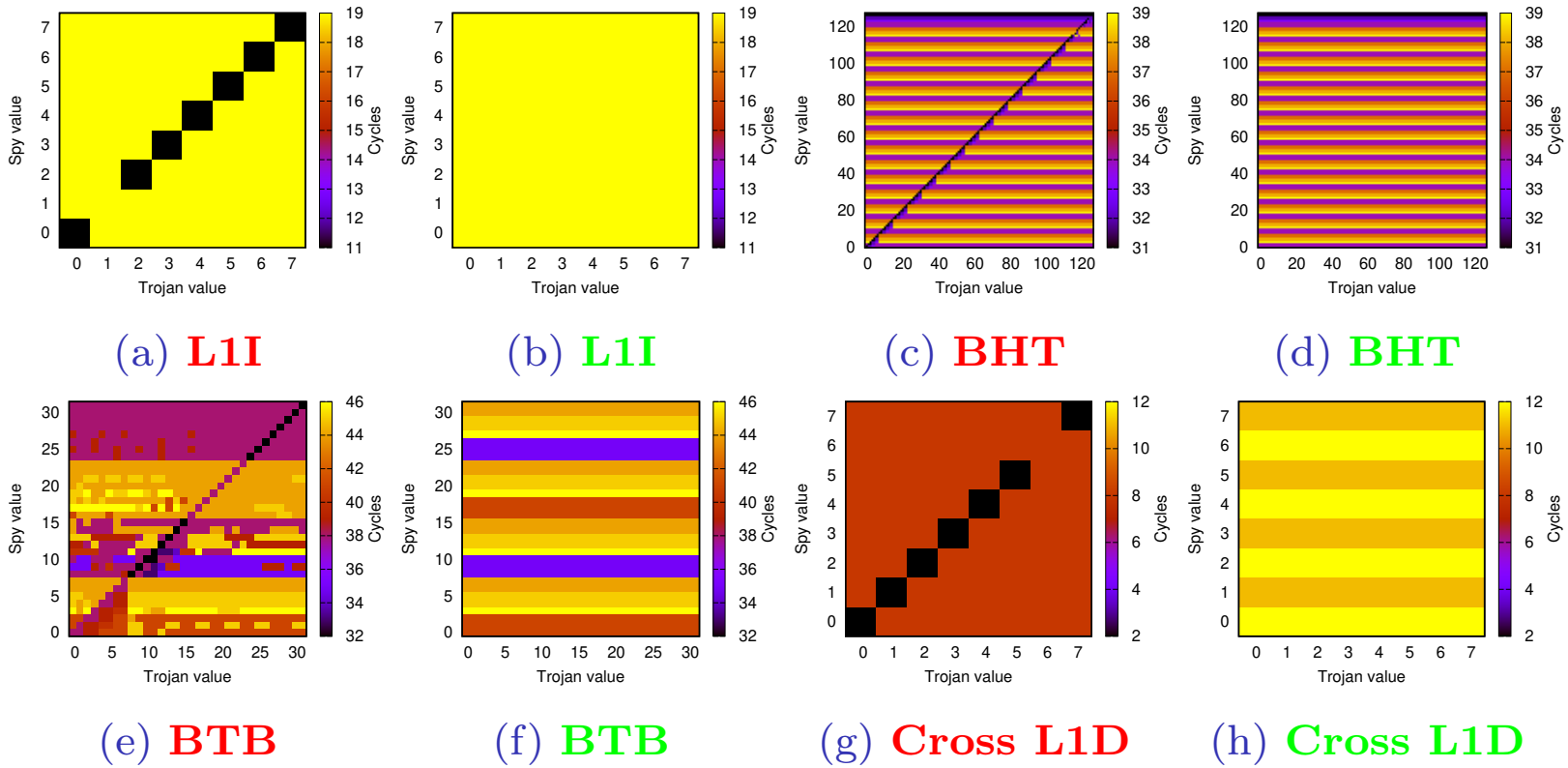


(a) Unprotected L1D



(b) Protected L1D

Other benchmarks



(a) **L1I** (b) **L1I** (c) **BHT** (d) **BHT**

(e) **BTB** (f) **BTB** (g) **Cross L1D** (h) **Cross L1D**

More under development ...

Table of contents

1 An isolation issue

2 ISA contextualization

3 Shared resources design

4 Timing evaluation

5 Conclusion

Conclusion

- Shared resources are sources of vulnerability.
- The ISA must be modified to give security information to the hardware.
- Software indicates its constraints, hardware applies them.
- A new security benchmark to evaluate the implementations.

Timesecbench: https://gitlab.inria.fr/rlasherm/timesecbench